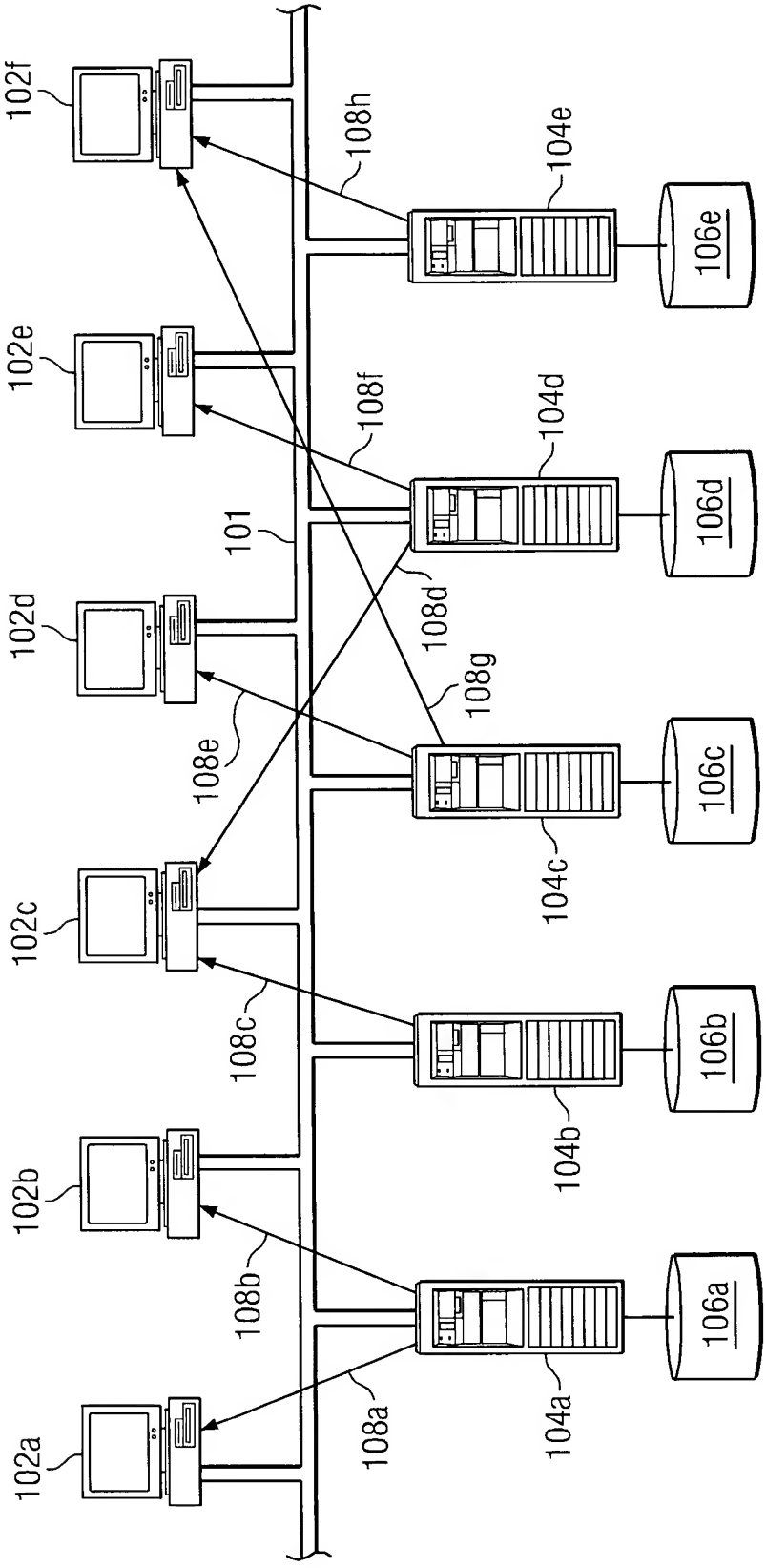
102f
102e
102d
102c
102b
102a
101
108h
108f
108d
108g
108e
108c
108b
108a
104e
104d
104c
104b
104a
106e
106d
106c
106b
106a

FIG. 1

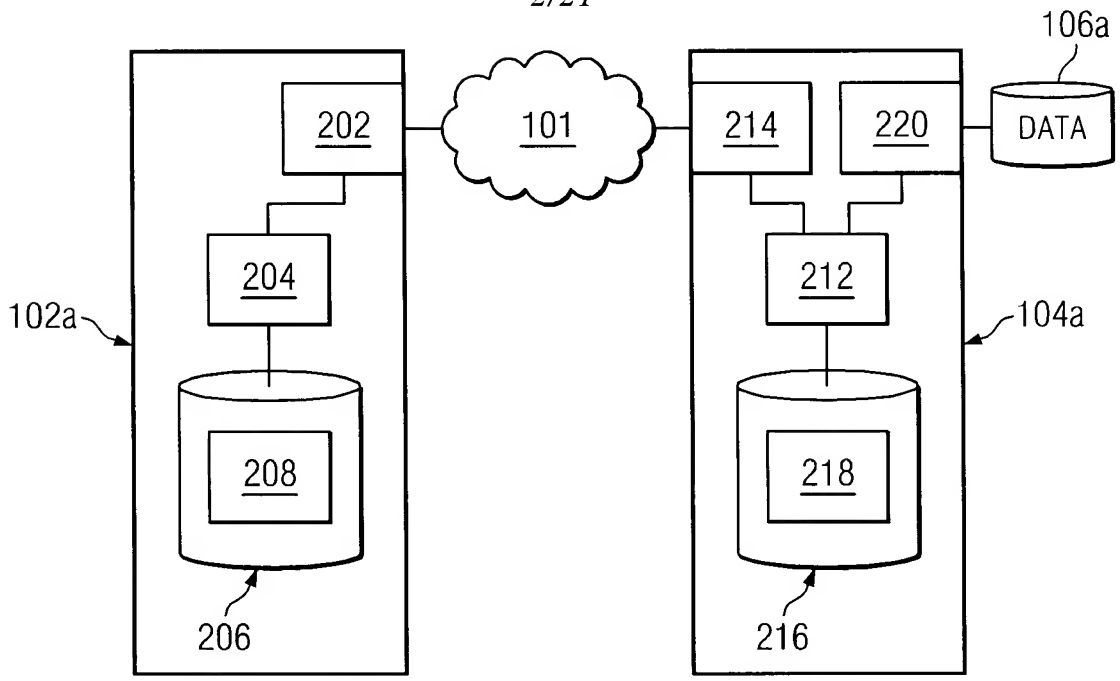

FIG. 2

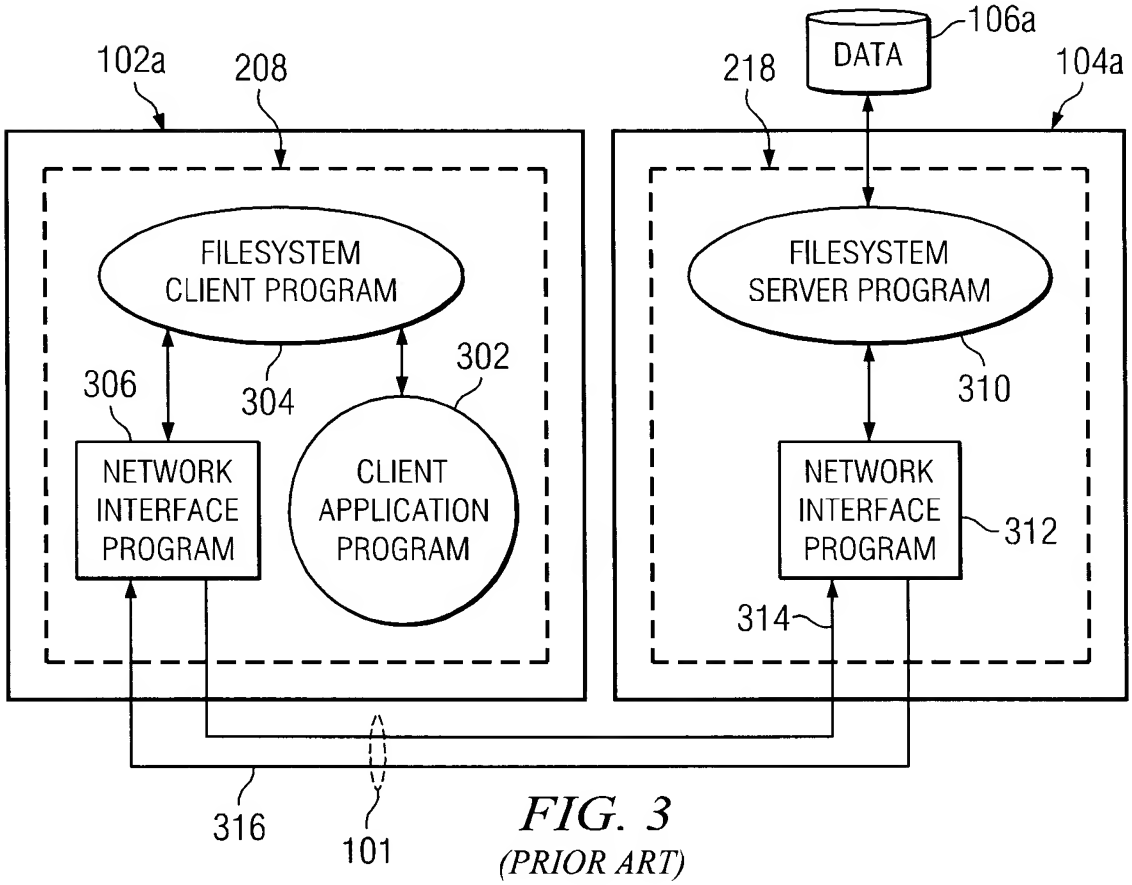

FIG. 3
(PRIOR ART)

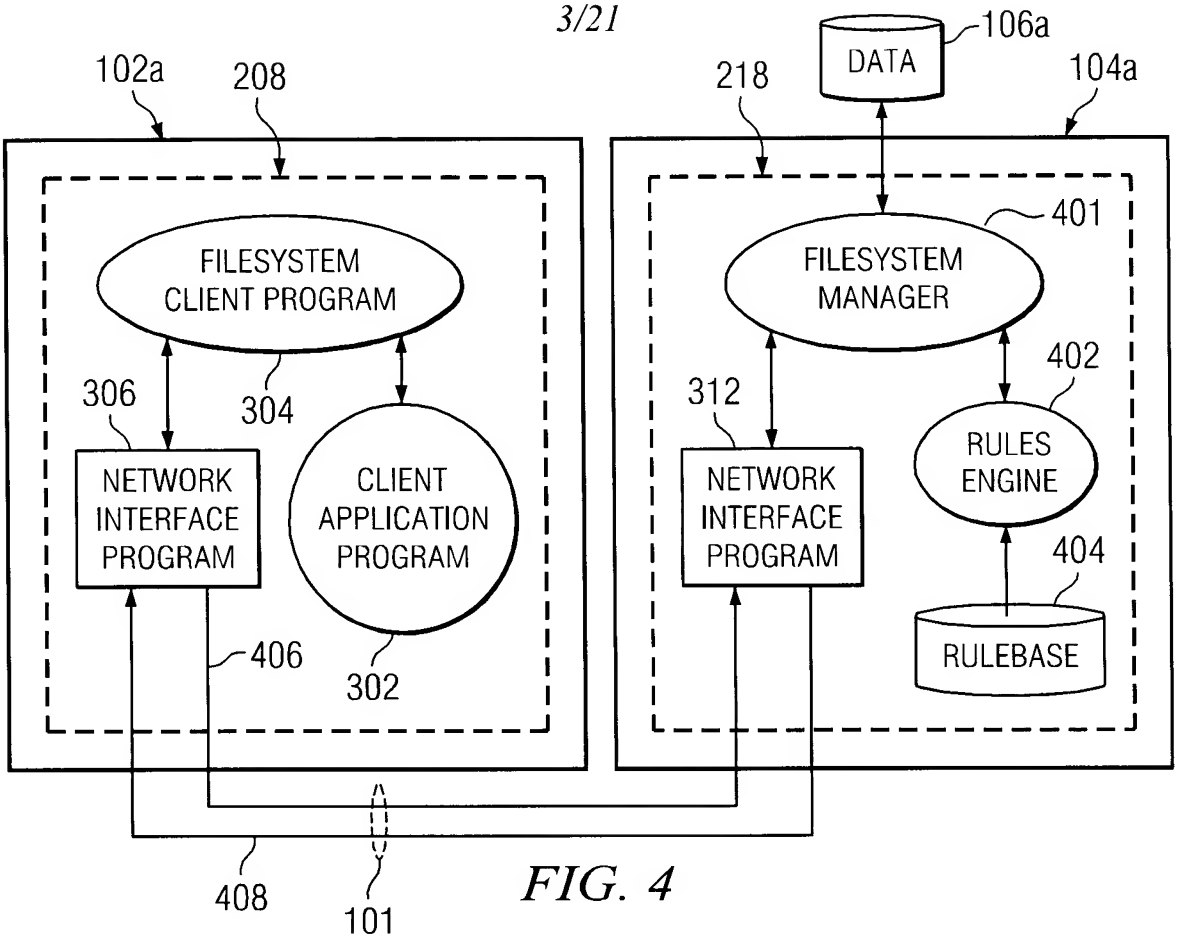
102a
208
218
DATA
106a
104a
FILESYSTEM
CLIENT PROGRAM
401
FILESYSTEM
MANAGER
306
304
312
402
NETWORK
INTERFACE
PROGRAM
CLIENT
APPLICATION
PROGRAM
NETWORK
INTERFACE
PROGRAM
RULES
ENGINE
404
406
302
RULEBASE
408
101

FIG. 4

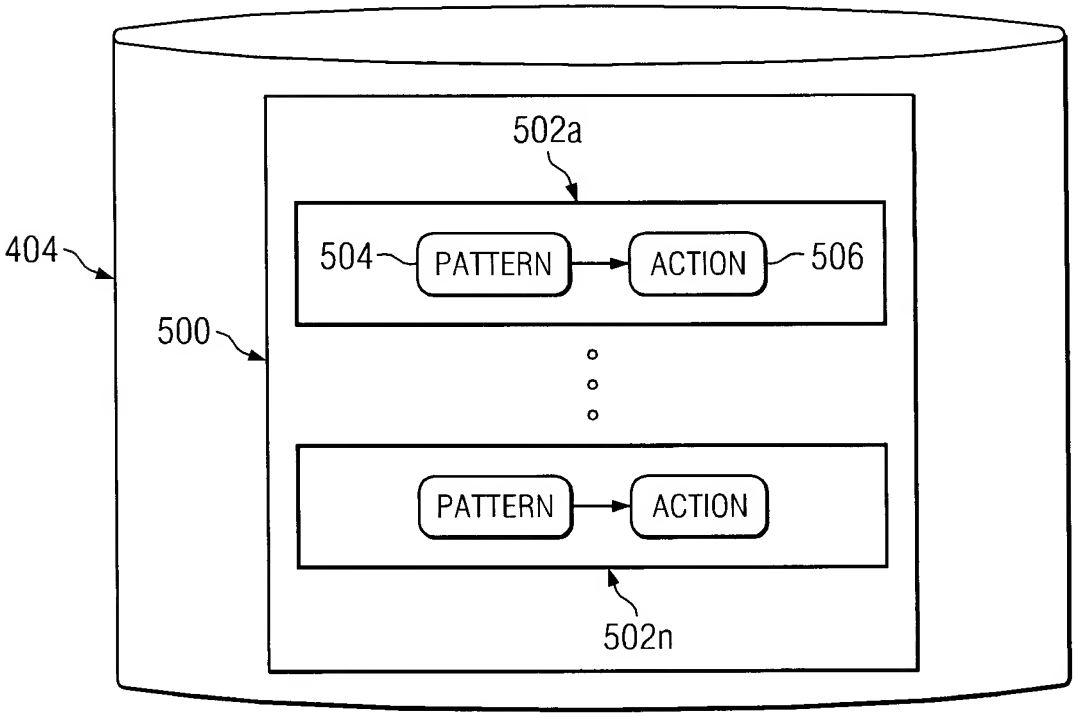
502a
404
504
PATTERN
ACTION
506
500
PATTERN
ACTION
502n

FIG. 5

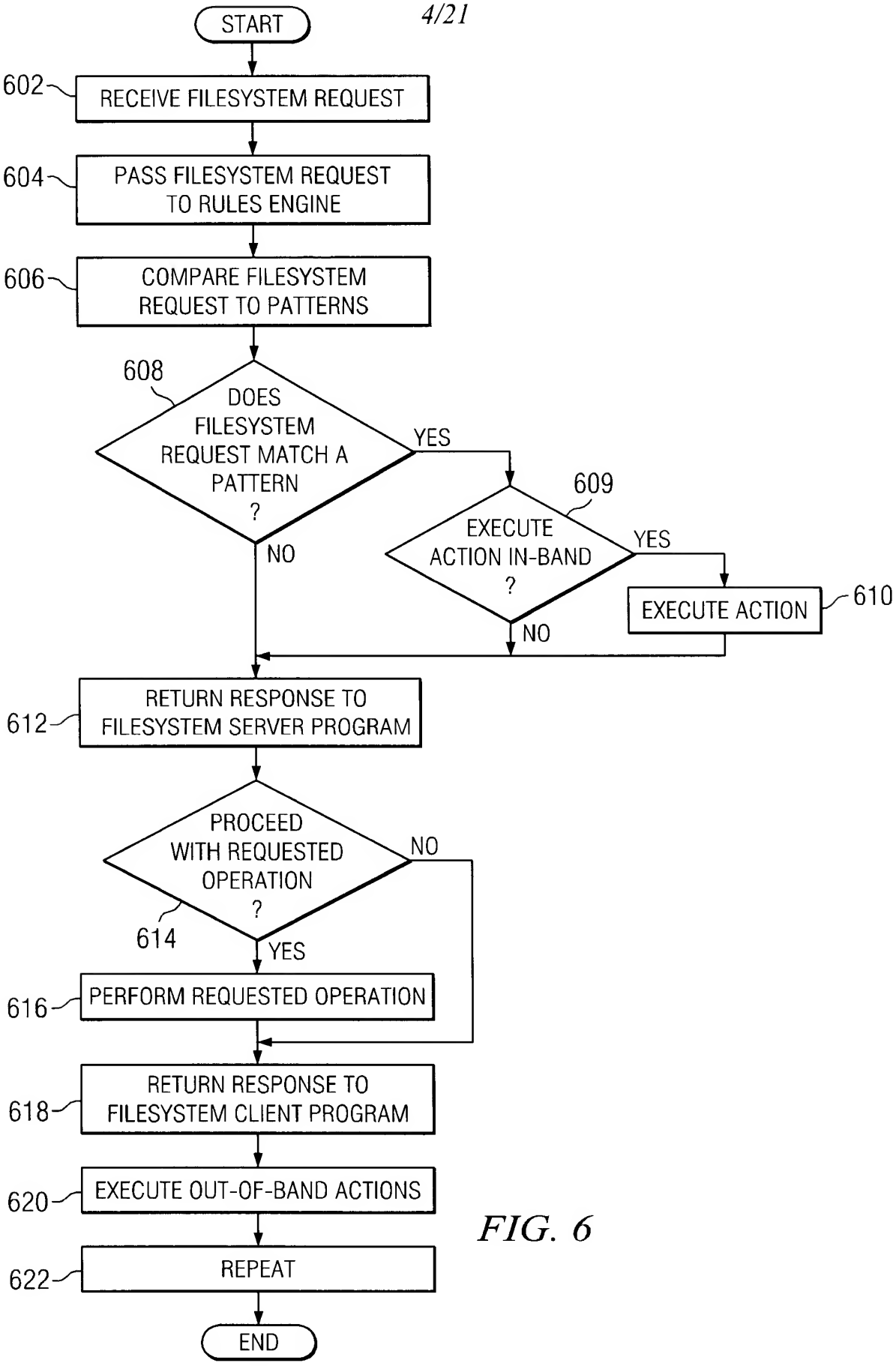
START
602
RECEIVE FILESYSTEM REQUEST
604
PASS FILESYSTEM REQUEST TO RULES ENGINE
606
COMPARE FILESYSTEM REQUEST TO PATTERNS
608
DOES FILESYSTEM REQUEST MATCH A PATTERN ?
YES
NO
609
EXECUTE ACTION IN-BAND ?
YES
NO
EXECUTE ACTION
610
612
RETURN RESPONSE TO FILESYSTEM SERVER PROGRAM
PROCEED WITH REQUESTED OPERATION ?
614
NO
YES
616
PERFORM REQUESTED OPERATION
618
RETURN RESPONSE TO FILESYSTEM CLIENT PROGRAM
620
EXECUTE OUT-OF-BAND ACTIONS
622
REPEAT
END

FIG. 6

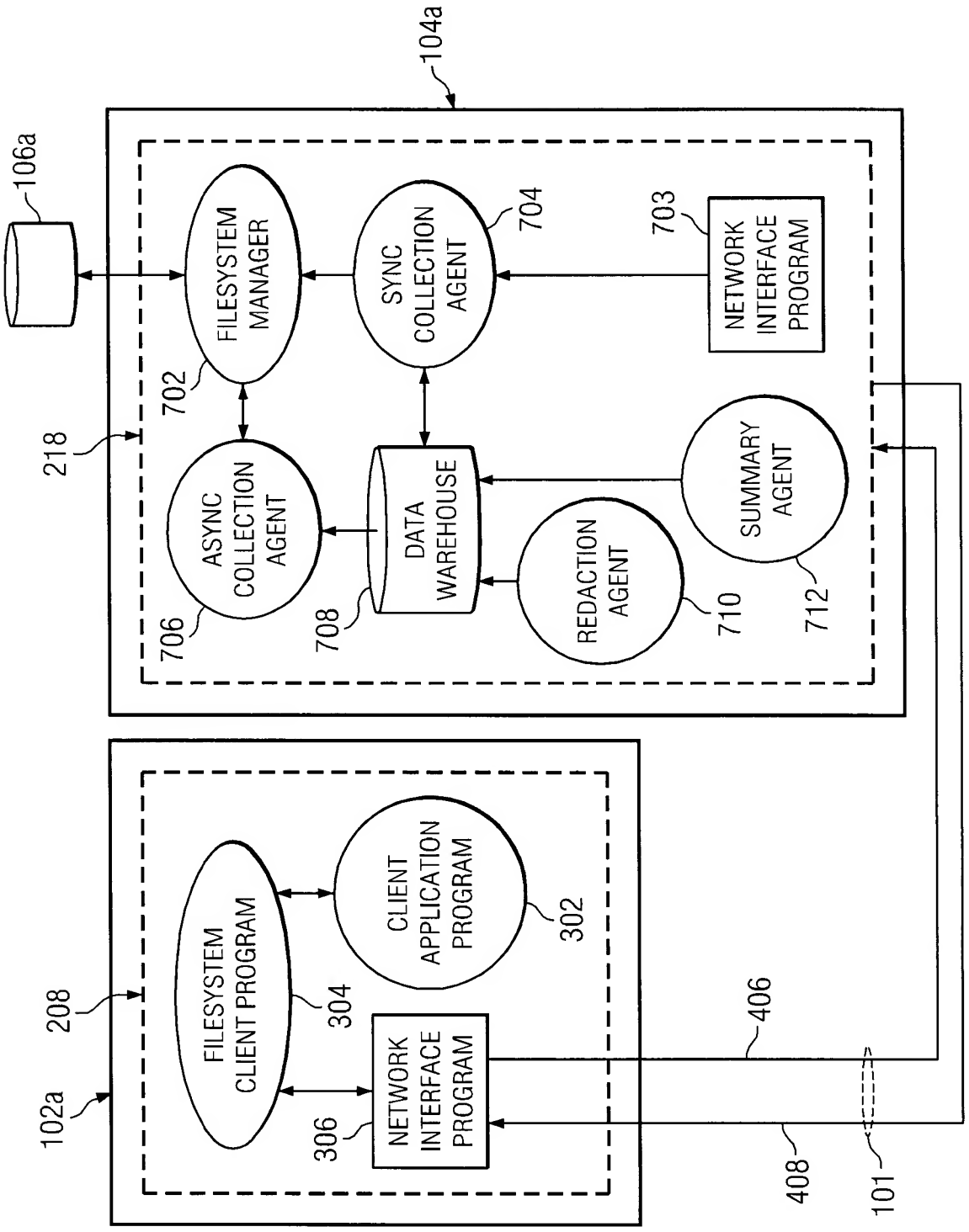
104a
106a
FILESYSTEM MANAGER
SYNC COLLECTION AGENT
NETWORK INTERFACE PROGRAM
ASYNC COLLECTION AGENT
DATA WAREHOUSE
REDACTION AGENT
SUMMARY AGENT
218
702
704
703
706
708
710
712
208
102a
FILESYSTEM CLIENT PROGRAM
CLIENT APPLICATION PROGRAM
NETWORK INTERFACE PROGRAM
304
302
306
406
408
101

FIG. 7

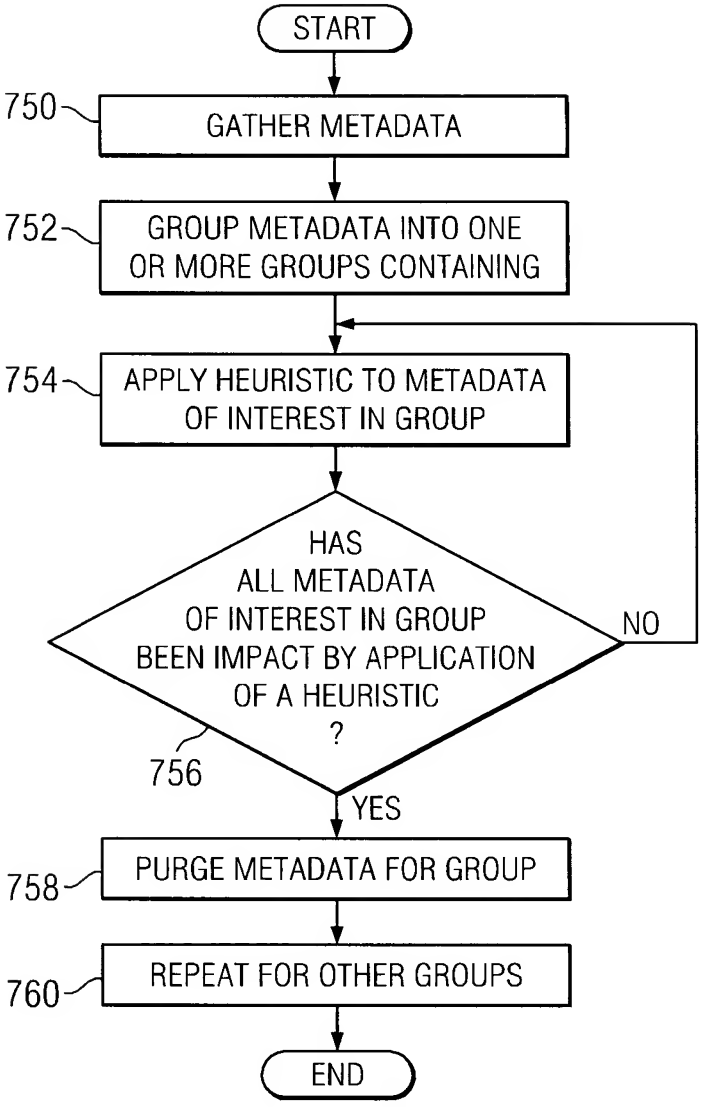
START
750
GATHER METADATA
752
GROUP METADATA INTO ONE OR MORE GROUPS CONTAINING
754
APPLY HEURISTIC TO METADATA OF INTEREST IN GROUP
HAS ALL METADATA OF INTEREST IN GROUP BEEN IMPACT BY APPLICATION OF A HEURISTIC ?
NO
756
YES
758
PURGE METADATA FOR GROUP
760
REPEAT FOR OTHER GROUPS
END

FIG. 8

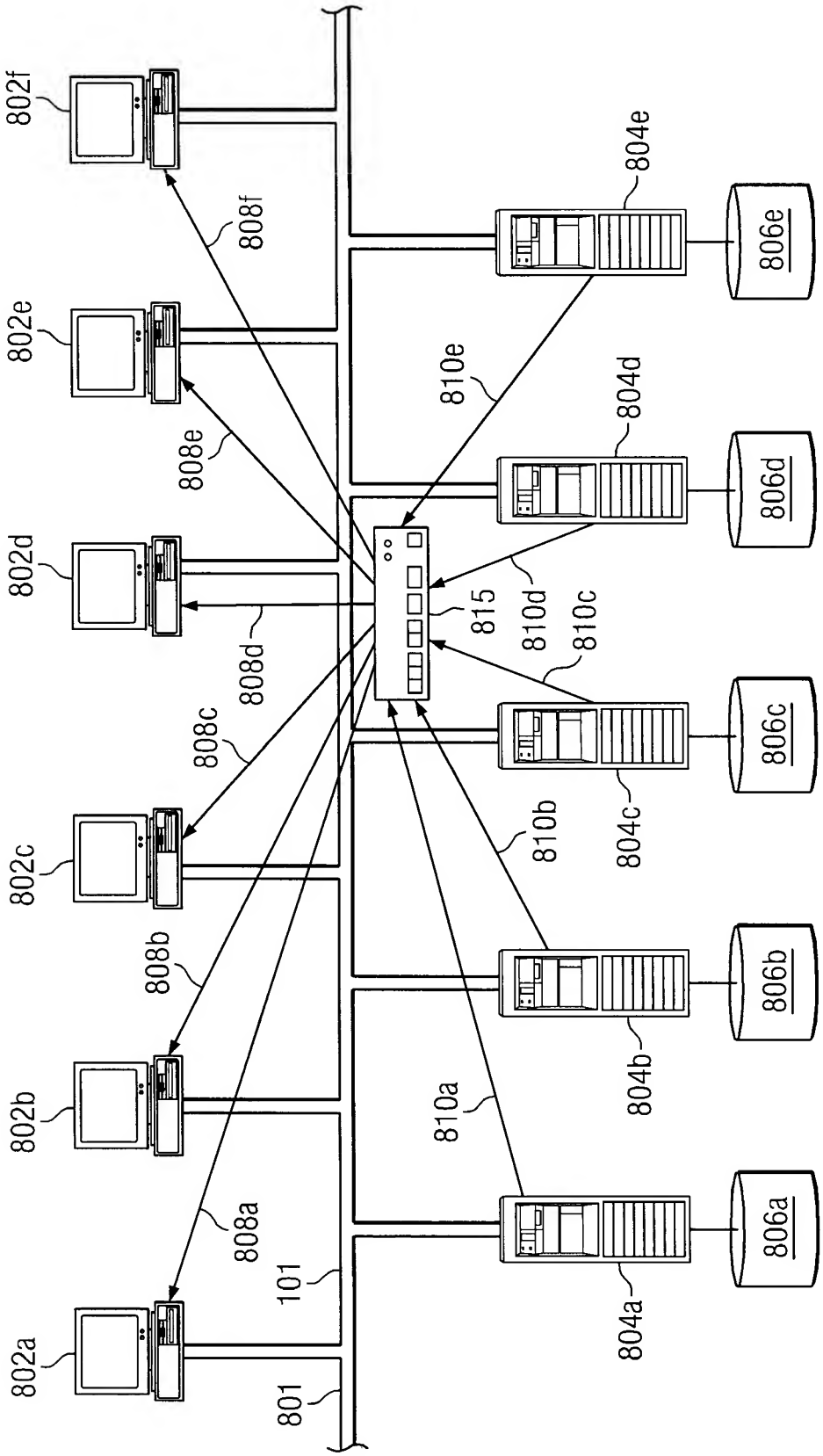
802f
802e
802d
802c
802b
802a
808f
808e
808d
808c
808b
808a
101
801
804e
804d
804c
804b
804a
806e
806d
806c
806b
806a
810e
810d
810c
810b
810a
815

FIG. 9

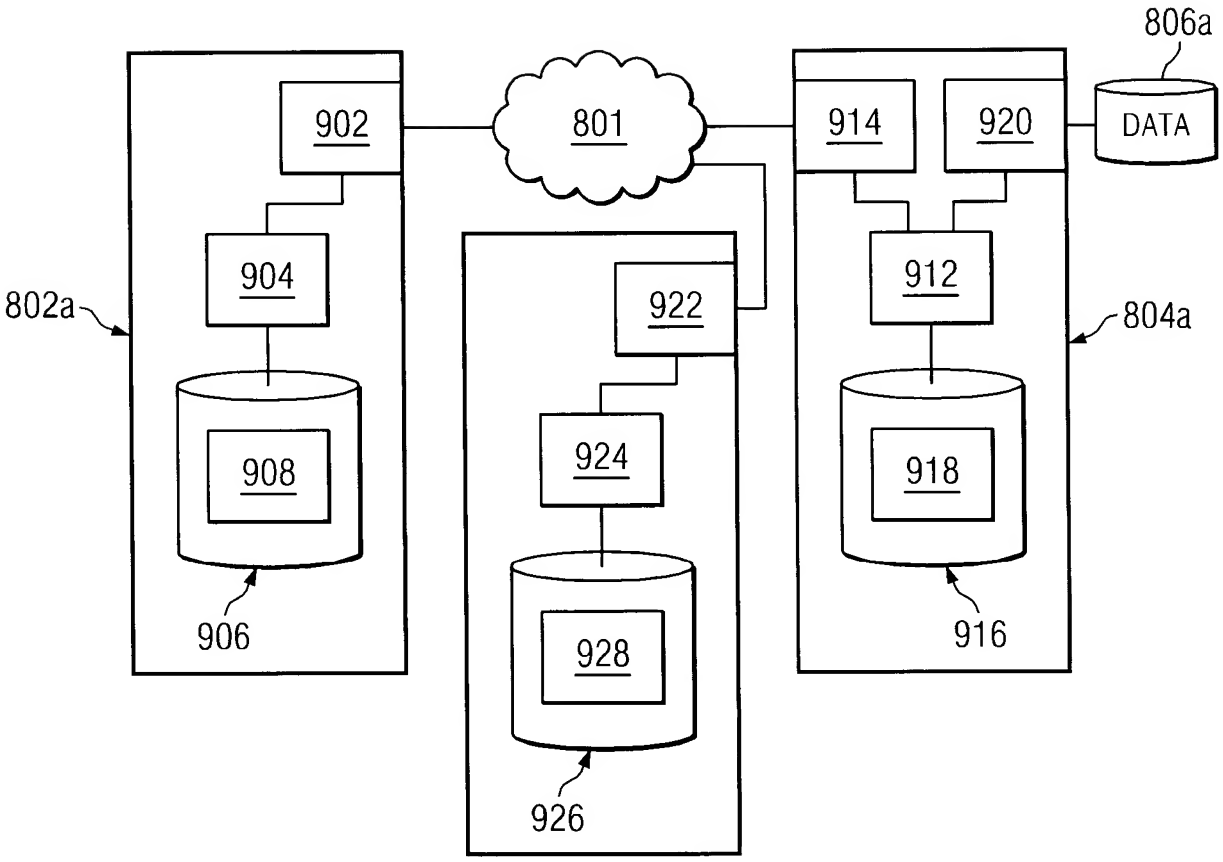
806a
902
801
914
920
DATA
904
912
802a
922
804a
908
924
918
906
916
928
926

FIG. 10

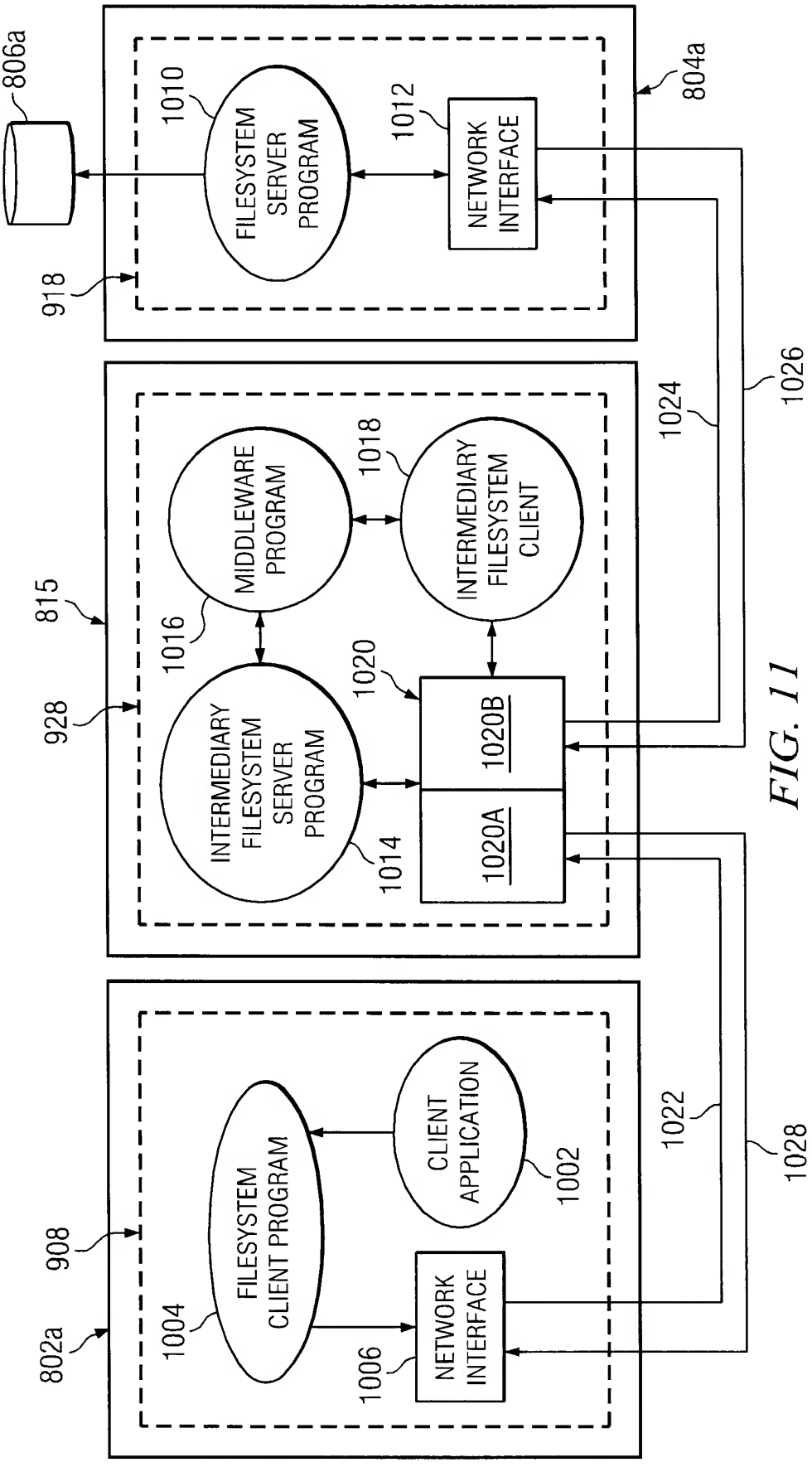
806a
804a
1010
FILESYSTEM SERVER PROGRAM
1012
NETWORK INTERFACE
918
1024
1026
815
MIDDLEWARE PROGRAM
1018
INTERMEDIARY FILESYSTEM CLIENT
1016
928
1020
1020B
1020A
INTERMEDIARY FILESYSTEM SERVER PROGRAM
1014
1022
1028
908
CLIENT APPLICATION
1002
802a
1004
FILESYSTEM CLIENT PROGRAM
1006
NETWORK INTERFACE

FIG. 11

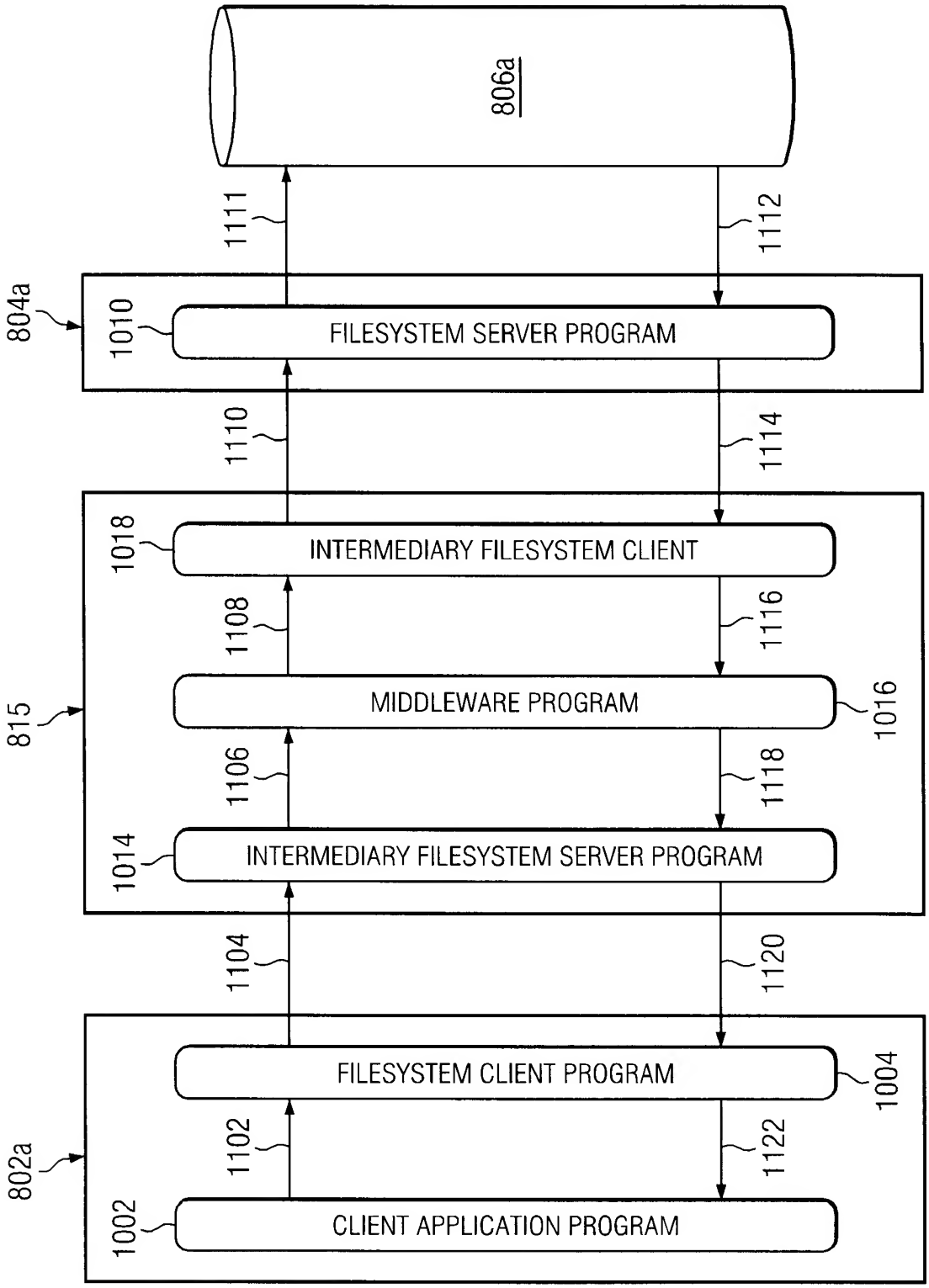
806a
1111
1112
804a
1010
FILESYSTEM SERVER PROGRAM
1110
1114
1018
INTERMEDIARY FILESYSTEM CLIENT
1108
1116
815
MIDDLEWARE PROGRAM
1016
1106
1118
1014
INTERMEDIARY FILESYSTEM SERVER PROGRAM
1104
1120
FILESYSTEM CLIENT PROGRAM
1004
1102
1122
802a
1002
CLIENT APPLICATION PROGRAM

FIG. 12

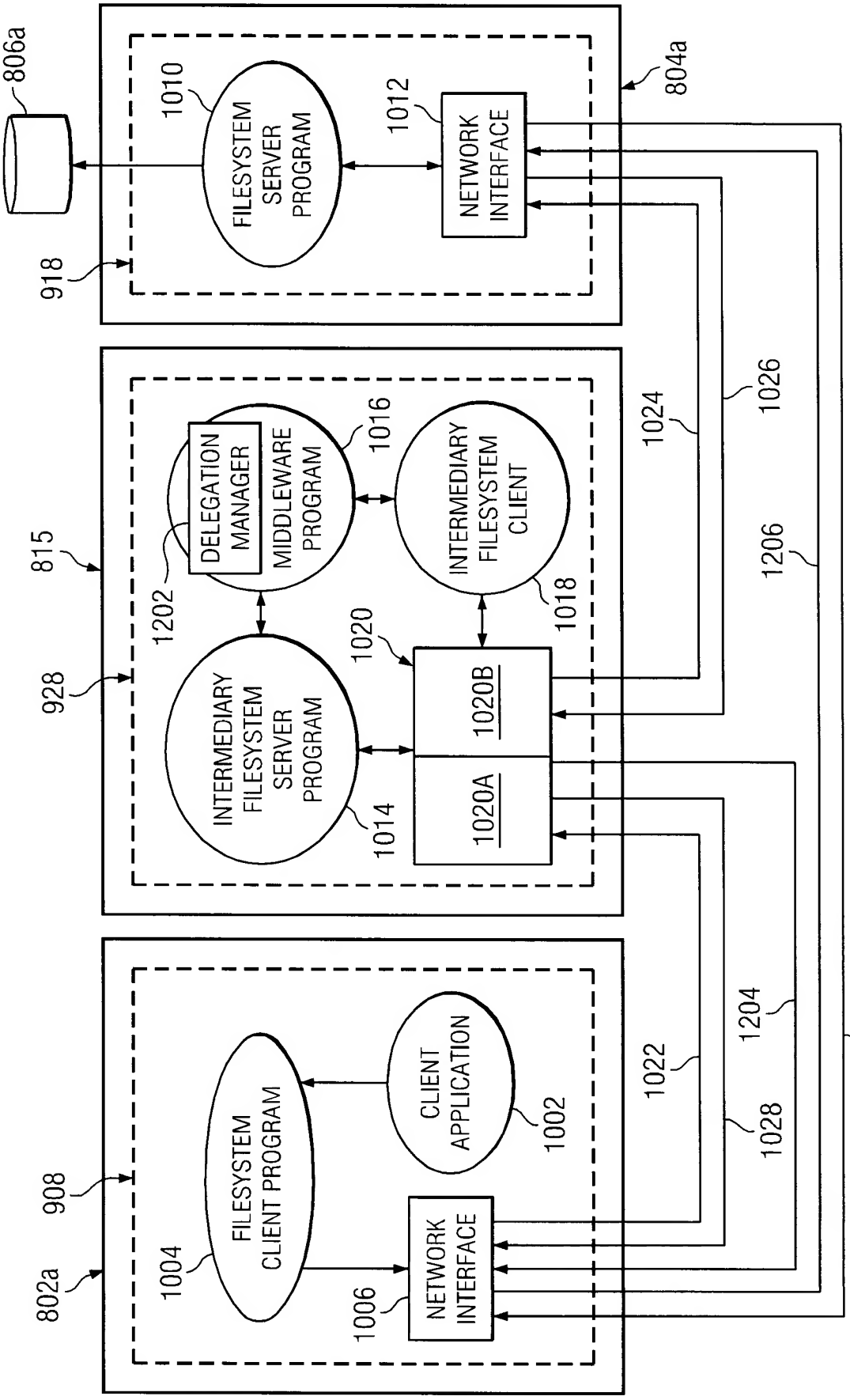
806a
804a
1010
FILESYSTEM SERVER PROGRAM
1012
NETWORK INTERFACE
918
815
928
DELEGATION MANAGER
MIDDLEWARE PROGRAM
1016
1202
INTERMEDIARY FILESYSTEM CLIENT
1018
1020
1020B
1020A
INTERMEDIARY FILESYSTEM SERVER PROGRAM
1014
1024
1026
1206
908
802a
FILESYSTEM CLIENT PROGRAM
1004
CLIENT APPLICATION
1002
1006
NETWORK INTERFACE
1022
1204
1028
1208

FIG. 13

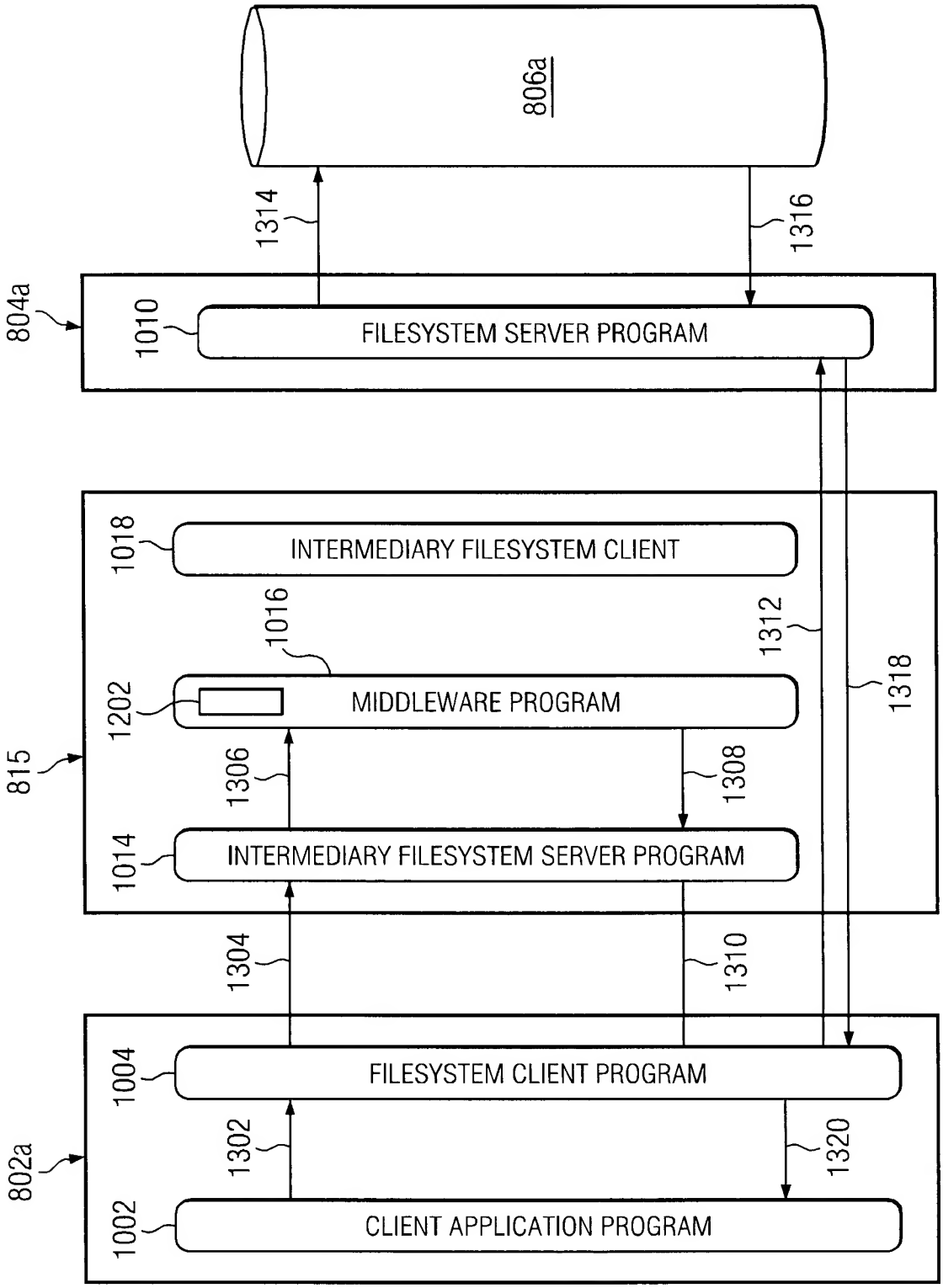
806a
1314
1316
804a
1010
FILESYSTEM SERVER PROGRAM
1018
INTERMEDIARY FILESYSTEM CLIENT
1016
1312
1318
1202
MIDDLEWARE PROGRAM
815
1306
1308
1014
INTERMEDIARY FILESYSTEM SERVER PROGRAM
1304
1310
1004
FILESYSTEM CLIENT PROGRAM
1302
1320
802a
1002
CLIENT APPLICATION PROGRAM

FIG. 14

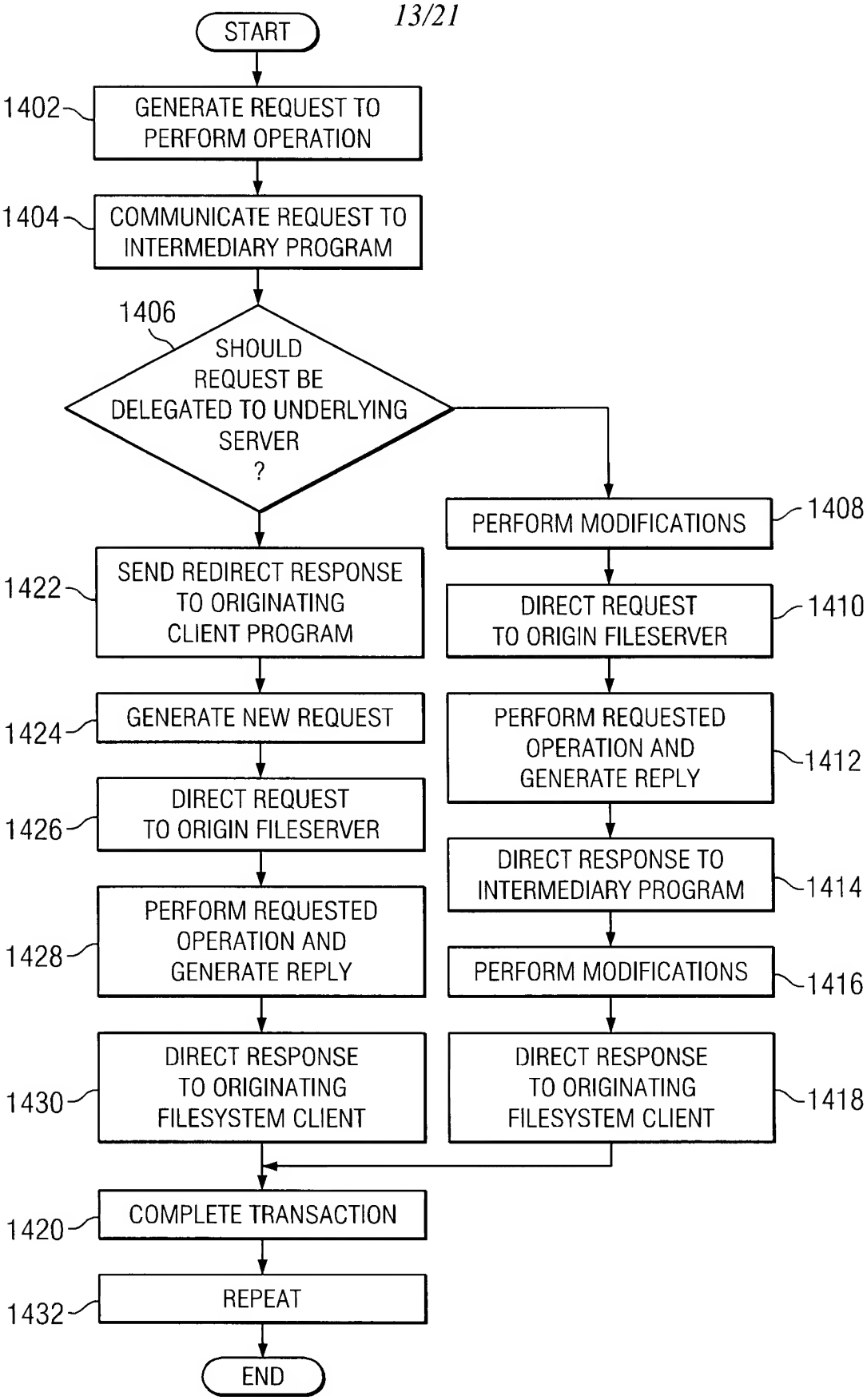
START
1402
GENERATE REQUEST TO PERFORM OPERATION
1404
COMMUNICATE REQUEST TO INTERMEDIARY PROGRAM
1406
SHOULD REQUEST BE DELEGATED TO UNDERLYING SERVER ?
1422
SEND REDIRECT RESPONSE TO ORIGINATING CLIENT PROGRAM
1424
GENERATE NEW REQUEST
1426
DIRECT REQUEST TO ORIGIN FILESERVER
1428
PERFORM REQUESTED OPERATION AND GENERATE REPLY
1430
DIRECT RESPONSE TO ORIGINATING FILESYSTEM CLIENT
1408
PERFORM MODIFICATIONS
1410
DIRECT REQUEST TO ORIGIN FILESERVER
1412
PERFORM REQUESTED OPERATION AND GENERATE REPLY
1414
DIRECT RESPONSE TO INTERMEDIARY PROGRAM
1416
PERFORM MODIFICATIONS
1418
DIRECT RESPONSE TO ORIGINATING FILESYSTEM CLIENT
1420
COMPLETE TRANSACTION
1432
REPEAT
END

FIG. 15

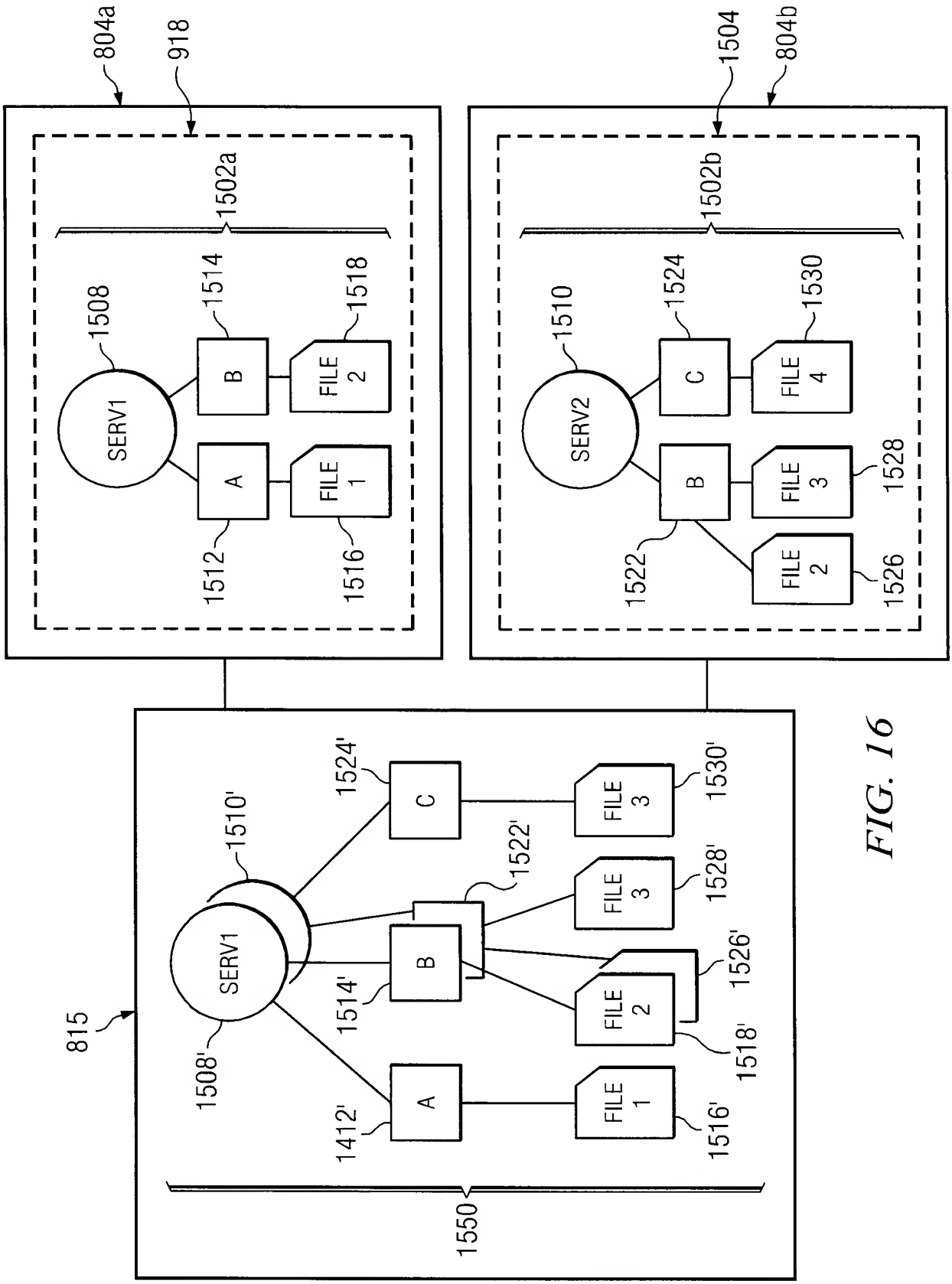
804a
918
1504
804b
1502a
1502b
1508
1514
1518
1510
1524
1530
SERV1
B
FILE 2
SERV2
C
FILE 4
A
FILE 1
B
FILE 3
1528
1512
1516
1522
FILE 2
1526
815
1510'
1524'
C
FILE 3
1530'
1522'
SERV1
1528'
FILE 3
B
1526'
1514'
FILE 2
1508'
1518'
1412'
A
FILE 1
1516'
1550

FIG. 16

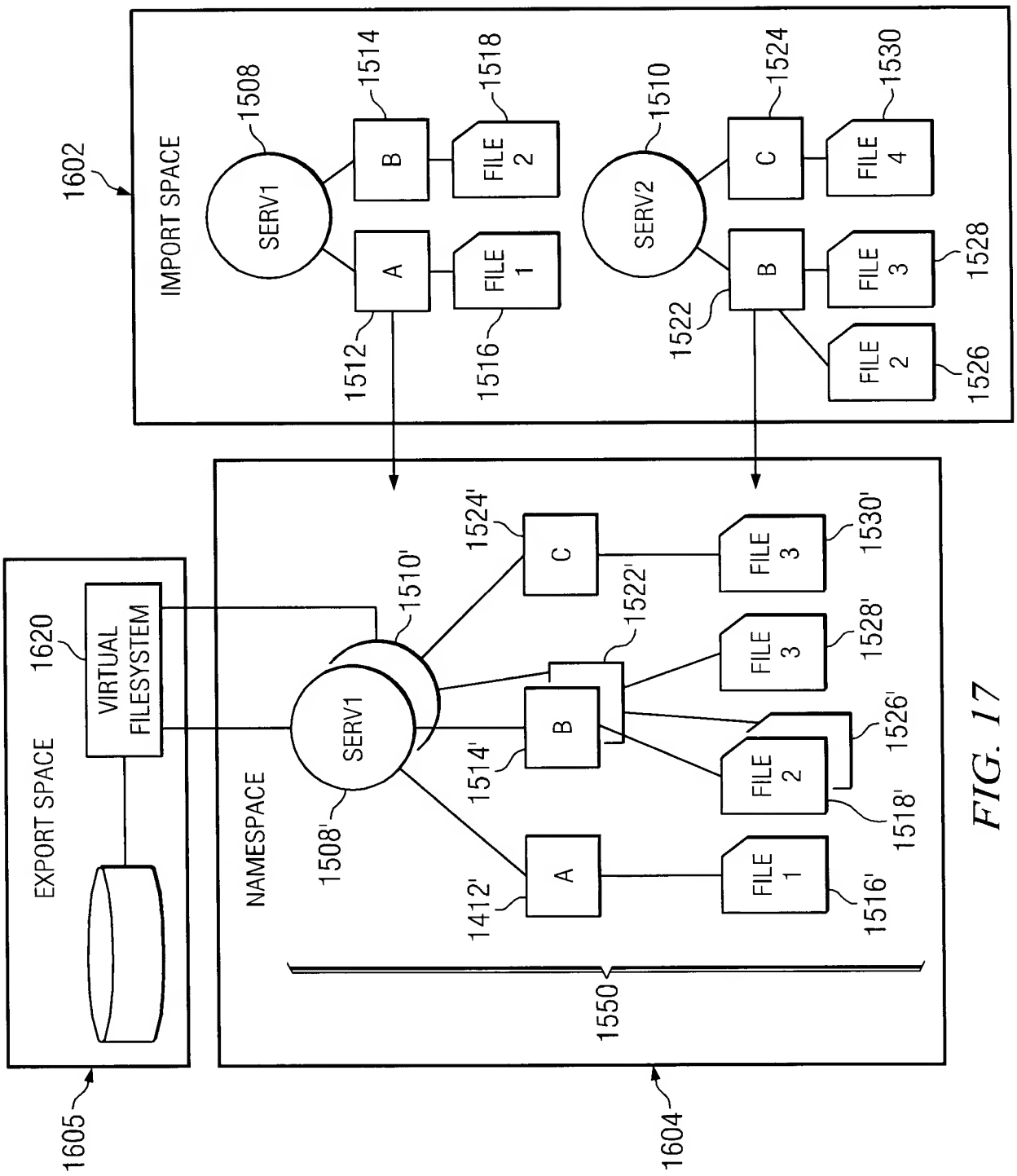
1602
IMPORT SPACE
SERV1
1508
B
1514
FILE 2
1518
A
FILE 1
1512
1516
SERV2
1510
C
1524
FILE 4
1530
B
FILE 3
1528
1522
FILE 2
1526
EXPORT SPACE
1620
VIRTUAL FILESYSTEM
NAMESPACE
SERV1
1508'
1510'
1524'
C
1522'
FILE 3
1530'
B
1514'
1528'
FILE 3
1526'
FILE 2
1518'
A
1412'
FILE 1
1516'
1550
1605
1604

FIG. 17

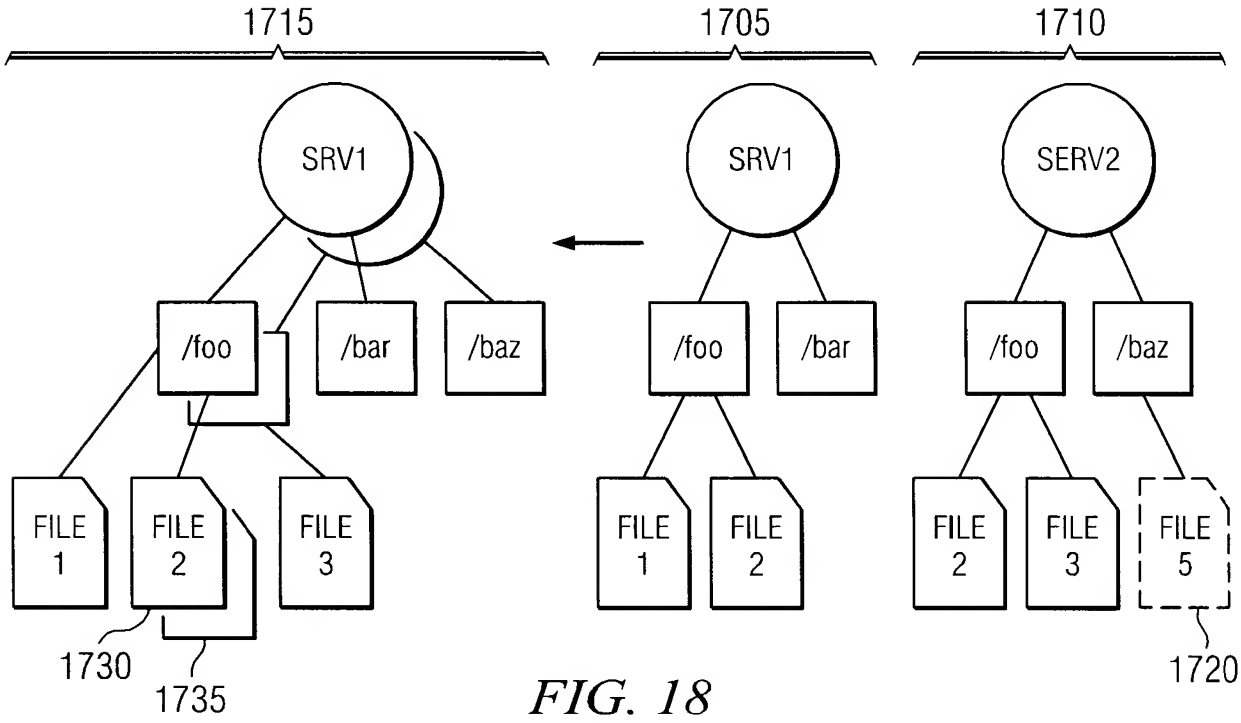
1715
1705
1710
SRV1
SRV1
SERV2
/foo
/bar
/baz
/foo
/bar
/foo
/baz
FILE 1
FILE 2
FILE 3
FILE 1
FILE 2
FILE 2
FILE 3
FILE 5
1730
1735
1720

FIG. 18

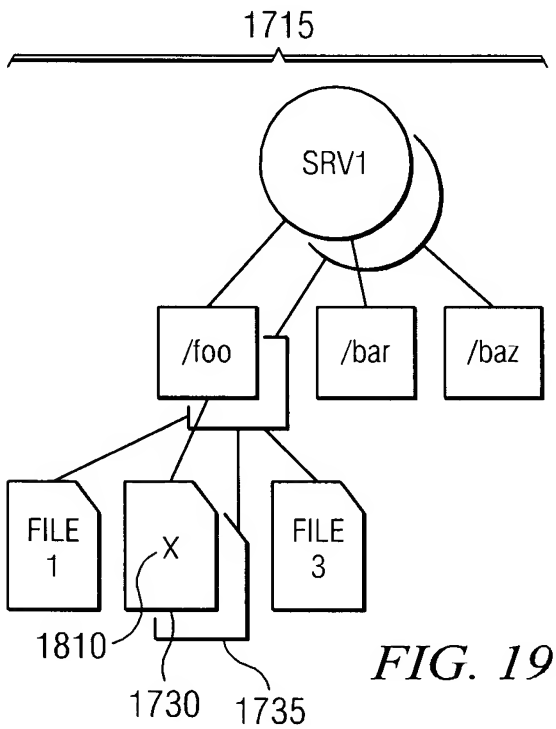
1715
SRV1
/foo
/bar
/baz
FILE 1
X
FILE 3
1810
1730
1735

FIG. 19

```
{{{
/* This module implements "write-through" semantics:
  First, the operation is attempted in the topbase,
  If the file/dir doesn't exist in the topbase, then
  it is attempted in the bottombases recursively.  We
  consider only pairwise layers; it is understood that
  the stack is arbitrarily deep, and upon each iteration
  through the stack the previous bottombase becomes the
  new topbase.

  Whiteouts:
  if a file exists in both layers:
    if it is removed: remove from top, create a whiteout to hide bottom
  if a file exits on top and not on bottom layer:
    if it is removed: remove from top
  if a file exists in bottom and not on top layer:
    if it is removed: remove from bottom
  if a file is whiteout on top and it exists in bottom:
  if a file is whiteout on top and it exists in bottom:
    if it is removed: do nothing
    if it is created: remove whiteout and create one on top
    if it is to be accessed: FAIL


  When an operation involves 2 file names:
  - rename(from,to) gets called only if from and to are in this namespace.
  - symlink(from,to) gets called only if from is in this namespace
          (to may or may not be in the name space)
  - link(from,to) gets called only if from and to are in this namespace
*/
```

FIG. 20

```
Operations on file that must exist.
getattr
readlink
chmod
chown
truncate
utime
read

Semantics:
fcn (path, args)
{
  GetTopPathState(path, NULL, &topExists, &isWhiteOut, &topPath);
  if (topExists)
  {
    // exists in top layer, use it
    return lowerFcn(topPath, args);
  }
  else
  {
    if (isWhiteOut)
      {
        // it's white out on top, FAIL
        return - ENOENT;
      }
      else
      {
        GetBottomPathState(path, NULL, &bottomExists, NULL, &bottomPath);
        if (bottomExists)
        {
          // doesn't exist on top, exists on bottom, use it
        return lowerFcn(bottomPath, args);
        }
        else
        {
          // doesn't exist on top or bottom
          return - ENOENT;
        }
      }
  }
 }
*/
```

FIG. 21

```
/* GROUP 2: ==========
  Operations on file that must not exist. Operation create the file.
  mknod
  mkdir

  Semantics:
  fcn (path, args)
  {
    GetTopPathState(path, &topMatchLen, &topExists, &isWhiteOut, &topPath);
    if (topExists)
    {
      // exists in top layer, FAIL
          return EEXIST;
    }
    else
    {
      if (isWhiteOut)
          {
            // it's white out on top, remove without and perform operation
            DelWhiteOut(topPath);
            return lowerFcn(topPath, args);
          }
          else
          {
            GetBottomPathState(path, &bottomMatchLen, &bottomExists, NULL, &bottomPath);
            if (bottomExists)
            {
              // exists on bottom, FAIL
              return EEXIST;
            }
            else
            {
              // doesn't exist on top or bottom, create file on layer with deeper match
              if (topMatchLen>= bottomMatchLen)
              {
                return lowerFcn(topPath, args);
              }
              else
              {
                return lowerFcn(bottomPath, args);
              }
            }
          }
      }
    }
  */
```

FIG. 22

```
/* GROUP 3: ==========
  Operations on file if it exists, file created if it doesn't.
  open
  write

  Semantics:
  fcn (path, args)
  {
    GetTopPathState(path, &topMatchLen, &topExists, &isWhiteOut, &topPath);
    if (topExists)
    {
      // exists in top layer, use it
          return lowerFcn(topPath, args);
    }
    else
    {
      if (isWhiteOut)
          {
            // it's white out on top, remove without and perform operation
            DelWhiteOut(topPath);
            return lowerFcn(topPath, args);
          }
          else
          {
            GetBottomPathState(path, &bottomMatchLen, &bottomExists, NULL, &bottomPath);
            if (bottomExists)
            {
              // exists on bottom, use it
              return lowerFcn(bottomPath, args);
            }
            else
            {
              // doesn't exist on top or bottom, create file on layer with deeper match
              if (topMatchLen>= bottomMatchLen)
              {
                return lowerFcn(topPath, args);
              }
              else
              {
                return lowerFcn(bottomPath, args);
              }
            }
          }
    }
  }
*/
```

FIG. 23

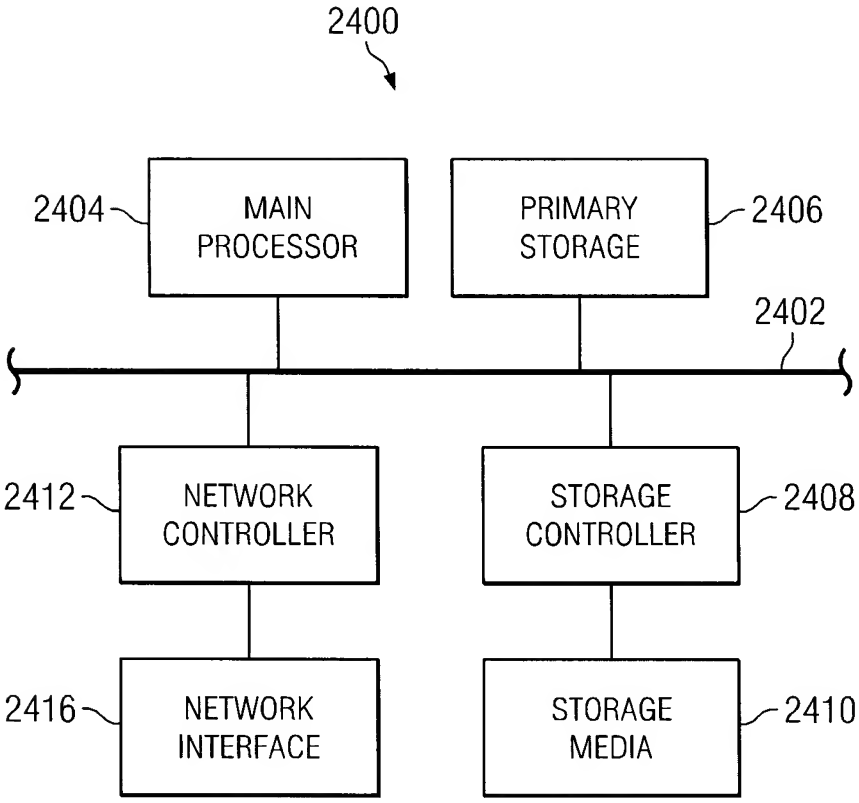
2400
2404
MAIN PROCESSOR
PRIMARY STORAGE
2406
2402
2412
NETWORK CONTROLLER
STORAGE CONTROLLER
2408
2416
NETWORK INTERFACE
STORAGE MEDIA
2410

FIG. 24